John West, Satheesh Kumar, Andy Kim

METHOD AND SYSTEM FOR RULE-BASED GENERATION OF AUTOMATION TEST SCRIPTS FROM ABSTRACT TEST CASE REPRESENTATION

BACKGROUND OF THE INVENTION

Field of the Invention:

[0001] This invention relates generally to systems and methods for generating test cases, and more particularly to methods and systems for generating test cases using rule-based generation of test cases from abstract representations.

Description of the Related Art:

[0002] Creation and maintenance of proper test cases that provide adequate coverage and effectiveness in terms of uncovering bugs is a very challenging and resource intensive activity. The current approaches in test case management do not ensure required level of reusability and maintainability of test cases. This results in repeating cycles of recreation of test cases from version to version, environment to environment and platform to platform.

[0003] Current approaches to test case creation are either manual programming in one of many programming languages or recording test cases using record/playback systems. In both approaches, the test cases are created and managed as sequence of programming language statements known as test scripts. These test scripts are then managed through a set of utilities which treats them like files on storage disks.

[0004] One of the disadvantages of managing test cases this way is that they are tied to a target execution environment. Different test execution environments support different scripting languages and same operation will be represented by different statements in these environments. If the enterprise software company decides to change the test execution system, all their test cases have to be recreated in the new environment.

[0005] Considering the substantial investments required by such an endeavor, many software development organizations will be tied to a single vendor of test execution environments. In addition, testing enterprise applications in customized environments also presents a challenge. Most of the enterprise applications are

substantially customized in their customer environments. Testing the application against a standard test suite can ensure that the application has not been altered in undesired ways. Providing such test cases can reduce the rollout time for such applications at customer sites as well as improve the efficiency of field support activities. The inability of customers to support the same test execution environments as that of the software provider makes this impossible.

[0006] Enterprise applications require testing across a large number of platforms. These platforms involve a technology stack of operating systems, database systems and other applications that they work with. For efficient management of test cases, the scripts shall not be tied to any specific platform.

[0007] Almost all enterprise applications support internationalization and localization, it is important that the same tests can be executed with data sets in different languages and different locale formats. In order to achieve this, the data sets have to be separated from the test scenarios. When test cases are managed as scripts, this requires one to parameterize the scripts manually. This again involves substantial investments of highly skilled labor.

[0008] The test cases that are parameterized, as described in the previous paragraph, are also required to ensure adequate coverage. The same test scenario must be executed with different datasets, different boundary condition values for example, to guarantee proper functioning of the system,

[0009] When an enterprise application moves from one release to the next, even subtle changes such as layout changes to user interfaces can result in changes to the test cases. Lack of abstractions within the test case representation using scripts substantially decreases the reusability of test cases and increases maintenance costs.

[0010] All enterprise applications have functionality that is repeated in many different contexts. The ability to create test cases in a modular fashion improves their reusability. The test cases for the same functionality may be reused in an test scenarios that uses the same functionality. Most scripting environments provide modularity through support of procedures. Such procedural abstractions are limited in their ability because the test cases not only encompass the procedural abstraction but also the data abstraction. Appropriate abstractions within the test case

management systems are to be devised so that test cases can be built in a modular fashion and recombined efficiently and effectively.

[0011] There is a need for improved systems and methods for generating test cases. There is a further need for improved methods and systems for generating test cases using rule-based generation of test cases from abstract representations.

SUMMARY OF THE INVENTION

[0012] Accordingly, an object of the present invention is to provide improved systems and methods for generating test cases.

[0013] Another object of the present invention is to provide improved methods and systems for generating test cases using rule-based generation of test cases from abstract representations.

[0014] Yet another object of the present invention is to provide methods and systems for generating test cases using rule-based generation of test cases from abstract representations that include application states, external interaction sequences and input data of test cases from data stores.

[0015] A further object of the present invention is to provide methods and systems for generating test cases using rule-based generation of test cases from abstract representations that include application states which represents a runtime snapshot of application under test which defines the context of external interaction.

[0016] Another object of the present invention is to provide methods and systems for generating test cases using rule-based generation of test cases from abstract representations that include application states that include a set of application objects, its attributes and attribute values.

[0017] These and other objects of the present invention are achieved in a method for generating test cases. Rule-based generation of test cases are provided from an abstract representation that includes application states, external interaction sequences and input data of test cases from data stores. Generated test cases are validated. The test cases are then converted to test scripts.

[0018] In another embodiment of the present invention, a computer system is provided that includes a processor coupled to a memory. The memory stores rule-based generation of test cases from an abstract representation that includes application states, external interaction sequences and input data of test cases from

data stores to produce test cases. Logic validates the test cases. Logic converts the test cases to test scripts.

BRIEF DESCRIPTIONS OF THE DRAWINGS

[0019] Figure 1 is a schematic diagram illustrating one embodiment of a test case transformation embodiment of the present invention.

[0020] Figure 2 is a general flowchart illustrating the Figure 1 embodiment.

[0021] Figure 3 is a schematic diagram illustrating the relationship of application states and interaction representation utilized in one embodiment of the present invention.

[0022] Figure 4 is a schematic diagram illustrating one embodiment of test case import utilized with the present invention.

[0023] Figure 5 is a flowchart illustrating one embodiment of import process utilized with the present invention.

[0024] Figure 6 is a schematic diagram illustrating an application object model that can be utilized with one embodiment of the present invention.

[0025] Figure 7 is a flowchart illustrating one embodiment of semantic analysis that can be utilized with the present invention.

[0026] Figure 8 is a schematic diagram illustrating a computer system that can be utilized to implement the Figure 1 test case transformation embodiment of the present invention.

[0027] Figure 9 is a schematic diagram illustrating one embodiment of a test case conversion embodiment of the present invention.

[0028] Figure 10 is a schematic diagram illustrating one embodiment of a test case conversion embodiment of the present invention utilizing composition, an abstract form and validation.

[0029] Figure 11 is a schematic diagram illustrating a computer system that can be utilized to implement the Figures 9 and 10 test conversion embodiments.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0030] Referring to Figures 1 and 2, one embodiment of the present invention is a system 10 and method for transforming test cases. Test cases 12 are imported that are written in one or more scripting languages. Test cases 12 are then

converted to an abstract representation 14 which includes one or more application states 16, external interaction sequences 18 and input data 20. Abstract representations 14 are stored in a database system 22. A variety of different database systems 22 can be utilized including but not limited to, a relational database management system, an XML database management system, and the like.

[0031] An application state 16 represents a runtime snapshot of an application under test which defines the context of external interaction. In one embodiment, illustrated in Figure 3, application state 16 is a set of application objects 24, such as a web page or a window control or an account object, for example. Each application object 24 is associated with a set of attributes 26 and their values. For example, a web page can have an attribute 26, called url, which contains the Uniform Resource Locator (URL) corresponding to the current web page, and an attribute 26, called title, which contains the title of the current web page. In one embodiment, the set of applications states 16 is represented in the test case 12 and are arranged in a hierarchical manner.

[0032] Scripting languages utilized can be typed or untyped programming languages used for recording or authoring test cases. External interaction sequences 18 can represent events invoked by external agents 28 on application objects 24. External agents 28 can be either human agents or other software agents. Interaction sequencing can include flow control structures 32 for capturing sequential, concurrent, looping, conditional interactions, and the like.

[0033] As shown in Figure 4, in one embodiment, a syntax analysis 34 can be implemented for incoming scripts. Syntax analyzer 34 can be implemented one for each scripting language. Syntax analyzer 34 can utilize rules of syntax analysis 36 that are specified in Extended Backus-Naur Form (EBNF). Syntax analysis can generate a parse tree in the form of an Abstract Syntax Tree (AST) 38. One embodiment of a method of handling scripts with the present invention is illustrated in the Figure 5 flowchart.

[0034] In one embodiment, a semantic analysis 40 is implemented that converts the AST 38 to an abstract internal test case representation 42 based on an Application Object Model (AOM) 44. Semantic analysis 40 decomposes the test cases represented as an AST 38 into application state 16, external interaction sequences and input data.

[0035] As illustrated in Figure 6, AOM 44 can be a metadata representation for modeling application under test. Components of the metadata representation include, but are not limited to, application object type definitions 48 for application objects 24, attribute definitions 50 for each application object 24 type, definitions of methods and events 52 that are supported by each application object 24 type, definitions of effects of events 52 on an application state 16, and the like. One embodiment of a method of performing semantic analysis with the present invention is illustrated in the Figure 7 flowchart.

[0036] Application object type definitions 48 can include additional categorization of each application object 24 type, and can be, (i) hierarchical, (ii) container and (iii) simple. The hierarchical object types are associated with an application state 16 of its own. Application object types 16 that can contain instances of other objects are container types. For example, a web page can be represented by a hierarchical object type and table within the web page by a container type. A label in the page is represented by a simple object type. The state associated with a hierarchical application object type 16 is a modal application state or a nonmodal application state. A modal application state restricts possible interactions to application object 24 instances available within a current application state 16. A dialog window for example restricts all user interactions to the current dialog window.

[0037] The effects of events 52 on an application state 16 capture one or more consequences of an event 52 to the application state 16. A consequence of an event 52 can be, creation of an instance of an object of a given type, deletion of an instance of an object type, modification of attributes of an existing object of type, selection of an instance of an object type, and the like.

[0038] Creation or selection of a hierarchical object type can result in formation of, a new application state 16, selection of the application state 16 associated with the object type, and the like.

[0039] In another embodiment, the abstract representation 14 of test cases 12 is enriched with information from an application metadata repository 54. The abstract representation 14 of test cases 12 can be enriched by extracting values for those attributes 26 of application objects 24 associated with the test cases 12 that are missing in the incoming test scripts. The enrichment of test cases 12 can

decouple test cases 12 and their recording or authoring environments, and the like, and allow usage of attributes 26 that are stable within an application metadata representation 54. For example, an internal identification field 56 within the application metadata repository 54 can be utilized to identify a given object 24 instead of a language dependent display label. This improves the reusability of the test case 12. Because different test execution environments can use different attributes 26 to identify the same application object 16, such decoupling provides platform independence.

[0040] In one embodiment, application object attributes 26 and input data are separated from external interaction sequencing to provide automatic parameterization. By automatically separating the data from the test case scenario, the 10 system dramatically reduces the manual labor involved to parameterize the scripts. Using the application object model, input data associated with each event 52 is separated from the scenario definition. The same process is applied to storing the object attributes 26. The input data definition forms a nested table data type definition that is driven for the events 52 involved in the scenario, and object event definitions in the application object model. This allows any data sets that match this definition to be applied to the same set of scenarios.

[0041] In another embodiment of the present invention, illustrated in Figure 8, a computer system 110 includes a processor 112 coupled to a memory 114. Memory 114 stores program instructions 116 executable by processor 112 for converting test cases to an abstract internal representation that includes application state, external interaction sequences and input data. A database 116 stores abstract representation of test cases. A syntax analyzer 118 can be included for incoming scripts. Syntax analyzer 118 generates a parse tree in the form of an Abstract Syntax Tree (AST) 120.

[0042] Logic 122 is provided to implement semantic analysis and convert AST 120 to an abstract internal test case representation 122 based on an Application Object Model (AOM). Logic 124 enriches the abstract internal test case representation with information from an application metadata repository 126. Logic 126 separates application object attributes and input data from external interaction sequencing to provide automatic parameterization.

[0043] Referring to Figures 1 and 2, one embodiment of the present invention is a system 10 and method for transforming test cases. Test cases 12 are imported that are written in one or more scripting languages. Test cases 12 are then converted to an abstract representation 14 which includes one or more application states 16, external interaction sequences 18 and input data 20. Abstract representations 14 are stored in a database system 22. A variety of different database systems 22 can be utilized including but not limited to, a relational database management system, an XML database management system, and the like.

[0044] An application state 16 represents a runtime snapshot of an application under test which defines the context of external interaction. In one embodiment, illustrated in Figure 3, application state 16 is a set of application objects 24, such as a web page or a window control or an account object, for example. Each application object 24 is associated with a set of attributes 26 and their values. For example, a web page can have an attribute 26, called url, which contains the Uniform Resource Locator (URL) corresponding to the current web page, and an attribute 26, called title, which contains the title of the current web page. In one embodiment, the set of applications states 16 is represented in the test case 12 and are arranged in a hierarchical manner.

[0045] Scripting languages utilized can be typed or untyped programming languages used for recording or authoring test cases. External interaction sequences 18 can represent events invoked by external agents 28 on application objects 24. External agents 28 can be either human agents or other software agents. Interaction sequencing can include flow control structures 32 for capturing sequential, concurrent, looping, conditional interactions, and the like.

[0046] As shown in Figure 4, in one embodiment, a syntax analysis 34 can be implemented for incoming scripts. Syntax analyzer 34 can be implemented one for each scripting language. Syntax analyzer 34 can utilize rules of syntax analysis 36 that are specified in Extended Backus-Naur Form (EBNF). Syntax analysis can generate a parse tree in the form of an Abstract Syntax Tree (AST) 38. One embodiment of a method of handling scripts with the present invention is illustrated in the Figure 5 flowchart.

[0047] In one embodiment, a semantic analysis 40 is implemented that converts the AST 38 to an abstract test case representation 42 based on an

Application Object Model (AOM) 44. Semantic analysis 40 decomposes the test cases represented as an AST 38 into application state 16, external interaction sequences and input data.

[0048] As illustrated in Figure 6, AOM 44 can be a metadata representation for modeling application under test. Components of the metadata representation include, but are not limited to, application object type definitions 48 for application objects 24, attribute definitions 50 for each application object 24 type, definitions of methods and events 52 that are supported by each application object 24 type, definitions of effects of events 52 on an application state 16, and the like. One embodiment of a method of performing semantic analysis with the present invention is illustrated in the Figure 7 flowchart.

[0049] Application object type definitions 48 can include additional categorization of each application object 24 type, and can be, (i) hierarchical, (ii) container and (iii) simple. The hierarchical object types are associated with an application state 16 of its own. Application object types 16 that can contain instances of other objects are container types. For example, a web page can be represented by a hierarchical object type and table within the web page by a container type. A label in the page is represented by a simple object type. The state associated with a hierarchical application object type 16 is a modal application state or a nonmodal application state. A modal application state restricts possible interactions to application object 24 instances available within a current application state 16. A dialog window for example restricts all user interactions to the current dialog window.

[0050] The effects of events 52 on an application state 16 capture one or more consequences of an event 52 to the application state 16. A consequence of an event 52 can be, creation of an instance of an object of a given type, deletion of an instance of an object type, modification of attributes of an existing object of type, selection of an instance of an object type, and the like.

[0051] Creation or selection of a hierarchical object type can result in formation of, a new application state 16, selection of the application state 16 associated with the object type, and the like.

[0052] In another embodiment, the abstract representation 14 of test cases 12 is enriched with information from an application metadata repository 54. The

abstract representation 14 of test cases 12 can be enriched by extracting values for those attributes 26 of application objects 24 associated with the test cases 12 that are missing in the incoming test scripts. The enrichment of test cases 12 can decouple test cases 12 and their recording or authoring environments, and the like, and allow usage of attributes 26 that are stable within an application metadata representation 54. For example, an identification field 56 within the application metadata repository 54 can be utilized to identify a given object 24 instead of a language dependent display label. This improves the reusability of the test case 12. Because different test execution environments can use different attributes 26 to identify the same application object 16, such decoupling provides platform independence.

[0053] In one embodiment, application object attributes 26 and input data are separated from external interaction sequencing to provide automatic parameterization. By automatically separating the data from the test case scenario, the 10 system dramatically reduces the manual labor involved to parameterize the scripts. Using the application object model, input data associated with each event 52 is separated from the scenario definition. The same process is applied to storing the object attributes 26. The input data definition forms a nested table data type definition that is driven for the events 52 involved in the scenario, and object event definitions in the application object model. This allows any data sets that match this definition to be applied to the same set of scenarios.

[0054] In another embodiment of the present invention, illustrated in Figure 8, a computer system 110 includes a processor 112 coupled to a memory 114. Memory 114 stores program instructions 116 executable by processor 112 for converting test cases to an abstract representation that includes application state, external interaction sequences and input data. A database 116 stores abstract representation of test cases. A syntax analyzer 118 can be included for incoming scripts. Syntax analyzer 118 generates a parse tree in the form of an Abstract Syntax Tree (AST) 120.

[0055] Logic 122 is provided to implement semantic analysis and convert AST 120 to an abstract test case representation 122 based on an Application Object Model (AOM). Logic 124 enriches the abstract test case representation with information from an application metadata repository 126. Logic 126 separates

application object attributes and input data from external interaction sequencing to provide automatic parameterization.

[0056] In another embodiment of the present invention, illustrated in Figures 9 and 10, methods and systems 210 are provided for generating test cases 212. The test cases 212 are generated from an abstract representation 214 that includes application states 216, external interaction sequences 218 and input data 220 of test cases from data stores 222. Test cases 212 are produced that are then validated. Test cases 212 are converted to test scripts 224. A variety of data stores 222 can be utilized including but not limited to, a relational database management system, an XML database management system, file system, and the like. Application states 216 can be the same as application states 26.

[0057] In one embodiment, rules 226 are provided for the selection of components of test case definition, namely application states 216, external interaction sequences 218 and input data 220, as well as rules for data driven test case generation 228. The selection rules 226 can be specified using query languages including but not limited to, SQL, Xquery, API called from code written in a programming language, and the like. The use of query languages allows test cases to be generated from live customer data.

[0058] In one embodiment of the present invention, generation of test case 212 includes composing the test case 212 as dictated by the input data 222. Multiple datasets 230 can be provided for at least a portion, or all, of the input data set 220 for a test case 212. This results in a generation of multiple test cases 212 or external interaction sequences repeated within a loop control structure for each dataset 230. Use of multiple datasets 230, for a portion of the input data 220, results in the interaction sequences corresponding to this portion of input data repeated within loop control structure such as a while loop.

[0059] In one embodiment, each element of input data 220 is flagged as valid or invalid using a data validity flag 232. The presence of a validity flag 232 in the input data 220, that is different from the one corresponding to the input data 220 when the test cases 212 were recorded or authored, results in the generation step including appropriate interaction sequences for exception handling. For example, a test case that was stored in the abstract representation 214 can have normal interaction sequence 218 when the valid input data sets 220 are provided. The

abstract representation also can contain interaction sequence 218 to be followed in the case of an exception condition such invalid data entry. The generator when generating the test case 212 from this abstract representation along with invalid input data will create a test case which includes interaction sequence 218 for exceptional situation rather than the normal interaction interaction sequence.

[0060] The generated test cases 212 can be validated against an external application meta data repository 238. The behavior of the validation can be controlled through additional validation rules 240.

[0061] The conversion of test cases 212 from an internal representation to a scripting language can be through platform specific mapping 234. The platform specific mappings include language mappings and other environment mappings. The language mapping used can map external interactions 218, captured as events on an application object, to appropriate statements in the scripting language 236. More than one language mapping can be provided at the same time. This allows generation of test scripts for multiple test execution environments. Additional environment mappings are provided to support additional platform independence. For example, if an application under test uses a third party report writer, the test cases can be represented using a generic report writer object and mappings for the specific report writer can be provided through the environment maps. This level of support can be extended to any level of underlying platform.

[0062] In another embodiment of the present invention, as illustrated in Figure 11 a computer system 310 is provided that includes a processor 312 and a memory 314 coupled to processor 310. Memory 314 stores rule-based generation of test cases 316 from an abstract representation 318. Abstract representation 318 includes application states, external interaction sequences and input data of test cases from data stores to produce test cases. Logic 320 validates the test cases. Logic 322 is provided for converting the test cases to test scripts.

[0063] Logic 320 provides that components of a test case definition, namely application states, external interaction sequences and input data, are consistent with each other and with an application object model. Logic 320 can be external validation logic. The external validation logic can include steps that validate a generated test case against an application metadata repository.

[0064] Computer system 310 can also include logic 324 that provides rules for selection of components of test case definition, namely application states, external interaction sequences and input data; rules for data driven test case generation. Computer system 310 can also include logic 326 that provides data driven test case generation. Logic 326 can compose the test case as dictated by the input data.

[0065] While embodiments of the invention have been illustrated and described, it is not intended that these embodiments illustrate and describe all possible forms of the invention. Rather, the words used in the specification are words of description rather than limitation, and it is understood that various changes may be made without departing from the spirit and scope of the invention.

[0066] What is claimed is: